Chapter 2

Fundamentals of MATLAB Programming

Scientific Computing with MATLAB, 2nd Edition

CRC/Taylor & Francis Press

Chinese version by Tsinghua University Press

PPT by Wenbin Dong and Jun Peng, Northeastern University, PRC

Proofread by Dingyu Xue & YangQuan Chen

Chapter 2 Fundamentals of MATLAB Programming

- Fundamentals of MATLAB Programming
- Fundamental Mathematical Calculations
- Flow Control Structures of MATLAB Language
- Writing and Debugging MATLAB Functions
- Two-, Three- Four-dimensional Graphics

Advantages of MATLAB

- MATLAB has the following advantages:
  - Clarity and high efficiency
  - Scientific computation, covers almost all the useful topics in math and engineering
  - Graphics facilities
  - Comprehensive toolboxes and block-sets, designed by experts of almost all disciplines
  - Powerful simulation facilities, unite the sub-systems of different domains together

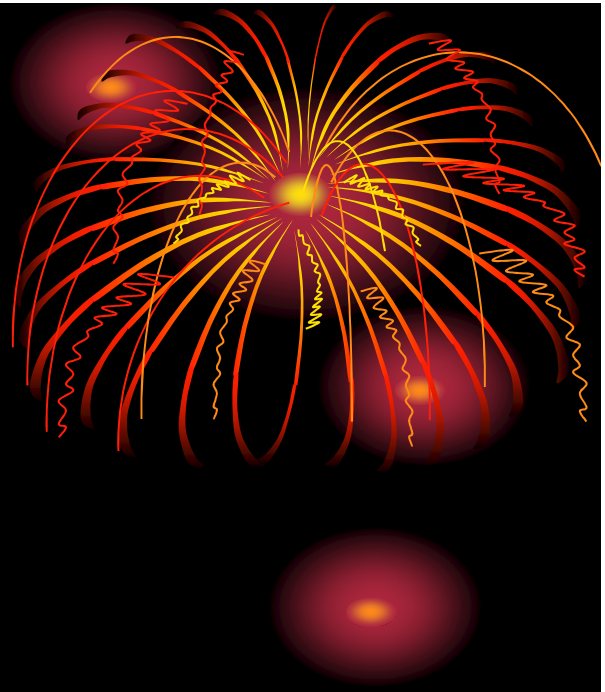
2.1 Fundamentals of MATLAB Programming

- Variables and constants in MATLAB
- Data structures
- Basic structure of MATLAB
- Colon expressions and sub-matrices extraction

2.1.1 Variables and Constants in MATLAB

- Variables in MATLAB
  - Starting with a letter followed by other characters:
  - Case-sensitive: **Abc ≠ABc**
  - Valid names: MYvar12, MY_Var12 and MyVar12_
  - Invalid variable names: 12MyVar, _MyVar12
- Constants in MATLAB:
  - **`eps, i, j, pi, NaN, Inf, i=sqrt(-1)`**
  - **`lastwarn, lasterr`**

2.1.2 Data structures



- Double-precision data type
- Symbolic data type
- Other data types

Double-precision Data Types

- IEEE standard, 64 bits (8 bytes)
  - 11 bits for exponential
  - 52 bits for numerical
  - 1 sign bit.

- Use double() to convert other types to double.
- Data range: 15 decimal digits

$$-1.7 \times 10^{308} \text{ to } 1.7 \times 10^{308}$$

- Other data types:
  - `int8(), int16(), int32(), uint16(), uint32()`

Symbolic Data Type

- Usually used in formula derivations and analytical solutions
  - variable declaration

```
syms var_list var_props
```

  - Data type conversion $B = \texttt{sym}(A)$
  - display the symbolic variables in any precision

$\texttt{vpa}(A)$, $\texttt{vpa}(A, n)$

✓The default value: 32 decimal digits.

More on Symbolic Variables

- More data types: integer
- Conditions specification
  - assume function

```
>> syms t x; assume(t<=4); assume(x~=2)
```

  - assumeAlso function

```
>> syms x real; assume(x>=-1);
   assumeAlso(x<5)
```

$$-1 \le x < 5$$

Example 2.1 Symbolic Variables

- Please declare a positive symbolic integer *k* such that it is a multiple of 13, and it does not exceed 3000
  - Not possible in earlier versions
  - How to declare 3000/13 ≈ 230.7

```
>> assume(k1<=230); assumeAlso(k1>0);
   assumeAlso(k1,'integer'); k=13*k1
```

Example 2.2 Value of π

- Display the first 300 digits of π.
- MATLAB command

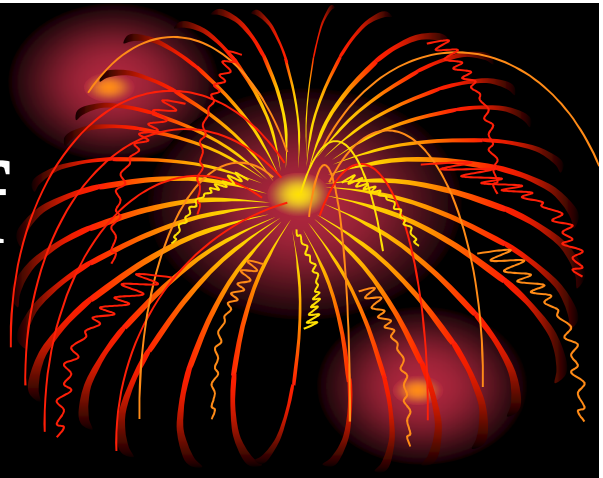
```
>> vpa(pi,300)
```

- One may further increase the number of digits to display.
- For extremely large number of digits, process may be slow

Other Data Types

- Strings: used to store messages. Single quotes
- Multi-dimensional arrays:
  - a direct extension of matrices with multiple indices.
- Cell arrays:
  - to put a set of data of different types under a single variable, expressed by { }.
- Classes and objects:
  - used in Object-Oriented Programming.

2.1.3 Basic Structures of MATLAB



- Direct assignment
  - The basic structure of this type of statement is

    `variable = expression`
  - A semicolon can prevent the results from display.
  - Reserved variable: `ans`, store the result of the latest statements without left-hand-variable
- Function call, returns many arguments

Example 2.3 Matrix Input

- Enter matrix into MATLAB

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

- Commands

```
>> A=[1,2,3; 4 5,6; 7,8 0]
```

- Other commands

```
>> A=[1,2,3; 4 5,6; 7,8 0];
   A=[[A; [1 2 3]], [1;2;3;4]];
```

Example 2.4 Complex Matrix

- Enter complex matrix into MATLAB

$$B = \begin{bmatrix} 1+j9 & 2+j8 & 3+j7 \\ 4+j6 & 5+j5 & 6+j4 \\ 7+j3 & 8+j2 & 0+j1 \end{bmatrix}$$

- MATLAB commands

```
>> B=[1+9i,2+8i,3+7i;
      4+6i 5+5i,6+4i; 7+3i,8+2i 1i]
```

- Things to avoid:

```
>> B=[1 +9i,2+8i,3+7j;
      4+6j 5+5i,6+4i; 7+3i,8+2j 1i]
```

Function Call Statements

- Function call statement

```
[returned_arguments]=
        function_name(input_arguments)
```

- Function call examples

```
[U  S  V]=svd(X)
```

- One function may be called in different ways
  - Built-in functions, *.m functions,
  - Anonymous functions, inline functions
  - Overload functions

2.1.4 Colon Expressions and Sub-matrices Extraction

- Colon expression is an effective way in defining row vectors.

$$v = s_1 : s_2 : s_3$$

- Start value s_1, increment s_2 and final value s_3
  - default increment: 1, when s_2 is missing
  - Value of s_3 not necessarily included in v

Example 2.5 Make Vectors

- For different increments, establish vectors for $t \in [0, \pi]$

```
>> v1=0: 0.2: pi
```

```
>> v2=0: -0.1: pi
```

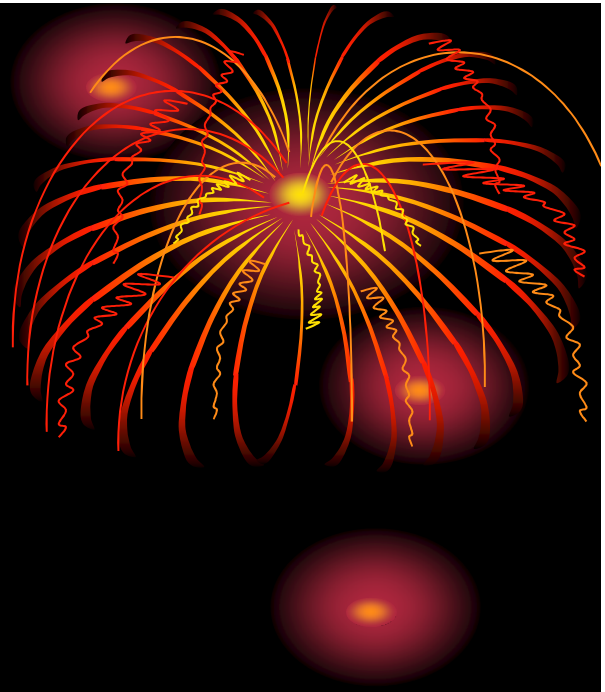
```
>> v3=0:pi
```

```
>> v4=pi:-1:0
```

- To include π

```
>> v5=linspace(0,pi,200);
```

Sub-matrix Extraction



- Basic format

$$B=A(v_1,v_2)$$

- Arguments
  - v_1 numbers of the rows
  - v_2 numbers of the columns
  - :, all the columns or rows, depending on the position of it
  - Key word end

Example 2.6 Sub Matrices

- Different sub-matrices can be extracted from the given matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

- MATLAB command

```
>> A=[1,2,3; 4,5,6; 7,8,0];
   B1=A(1:2:end, :)
   B2=A([3,2,1],[1 1 1])
   B3=A(:,end:-1:1)
```

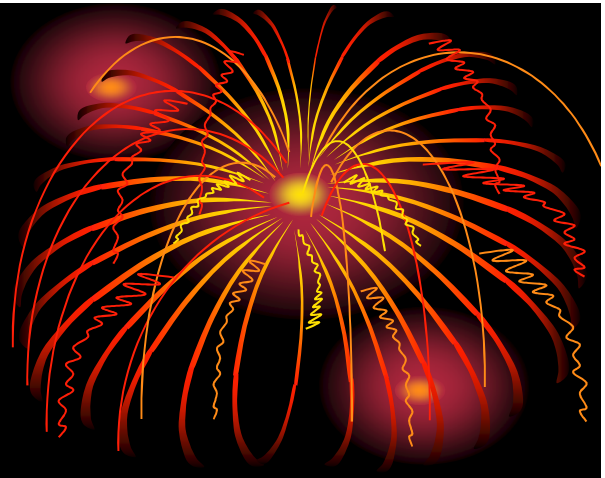
2.2 Fundamental Mathematical Calculations

- Algebraic operations of matrices
- Logic operations of matrices
- Relationship operations of matrices
- Simplifications and presentations of analytical results
- Basic number theory computations

2.2.1 Algebraic operations of matrices

- Matrix transpose
- Matrix addition and subtraction
- Matrix multiplications
- Matrix divisions
- Matrix flip and rotations
- Matrix power
- Matrix dot operations

Matrix transpose



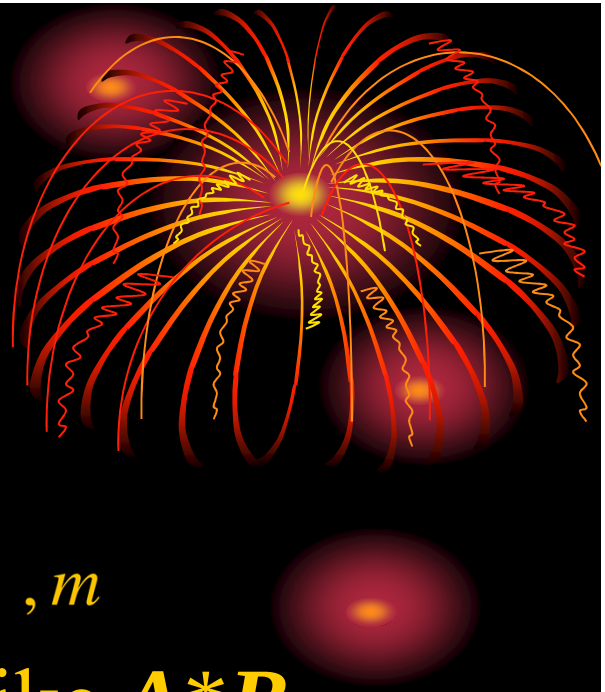
- Matrix representation:
  - Matrix $\boldsymbol{A}$, n rows and m columns, is referred to as an $n \times m$ matrix
- Hermitian transpose $C = A'$

$$C = A^{\mathrm{H}}, c_{ji} = a_{ij}^{*},\ i = 1, \cdots, n,\ j = 1, \cdots, m$$

- Simple transpose $C = A.'$

$$B = A^{\mathrm{T}}, b_{ji} = a_{ij},\ i = 1, \cdots, n,\ j = 1, \cdots, m$$

Matrix Addition and Subtraction



- Mathematical representations

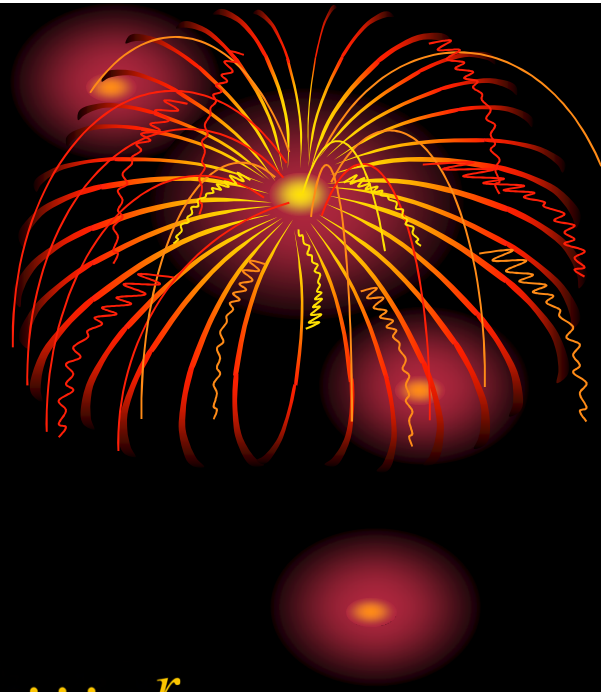
$$c_{ij} = a_{ij} \pm b_{ij},\ i = 1, \cdots, n,\ j = 1, \cdots, m$$

- Difficult to program under C, like ***A*B***
- MATLAB implementation

$$C = A + B \qquad C = A - B$$

- Note: either variable can be a scalar
- If not compatible, error messages given

Matrix Multiplication



- Math expression $C = AB$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

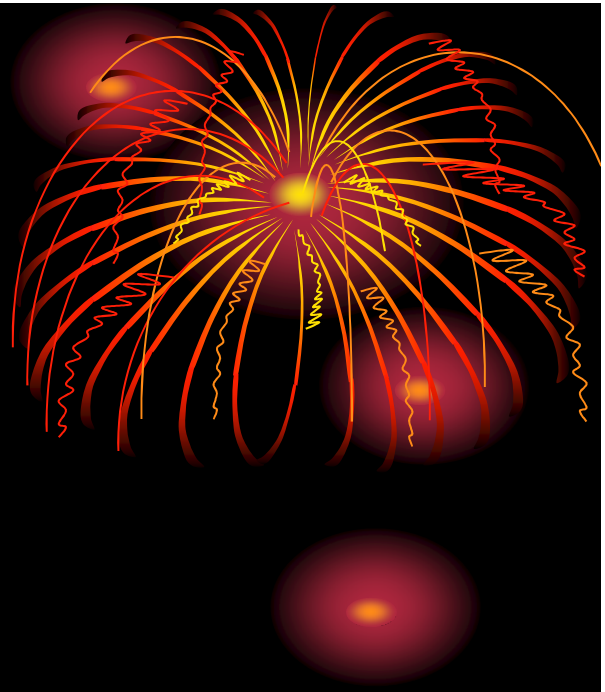
$$i = 1, 2, \cdots, n, \quad j = 1, 2, \cdots, r$$

- MATLAB expression

```
C=A*B
```

- Note: compatibility auto-checked
  - Applies to complex and others

Matrix Division



- Matrix left division
  - Solve the linear equations:

$$AX = B$$

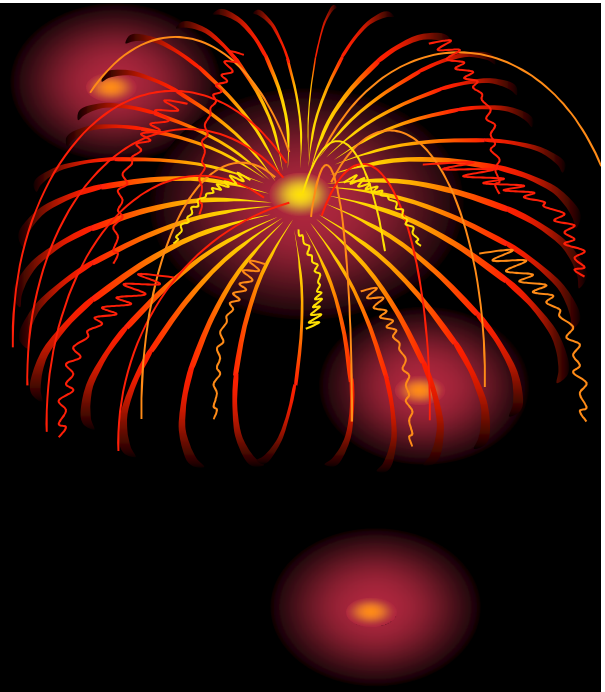
  - MATLAB solution:

$$X = A\backslash B$$

  - Least squares solution
  - If A is a non-singular square matrix. Then,

$$X = A^{-1}B$$

Matrix Right Division



- Mathematical expression XA=***B***
  - MATLAB solution

$$X = B/A$$

  - Least squares solution
  - If A is a nonsingular square matrix. Then,

$$X = BA^{-1}$$

  - More precisely,

$$B/A=(A'\backslash B')'$$

Matrix Flips and Rotations

- left-right flip

$$b_{ij} = a_{i,n+1-j} \qquad B = \texttt{fliplr}(A)$$

- up-down flip

$$c_{ij} = a_{m+1-i,j} \qquad B = \texttt{flipud}(A)$$

- Rotate 90^o , counterclockwise

$$d_{ij} = a_{j,n+1-i} \qquad D = \texttt{rot90}(A)$$

- How to rotate 180^o , 270^o ?

$$D = \texttt{rot90}(A,k)$$

Matrix Power

- $\boldsymbol{A}$ must be a square matrix
- Matrix $\boldsymbol{A}$ to the power x.
  - Math description

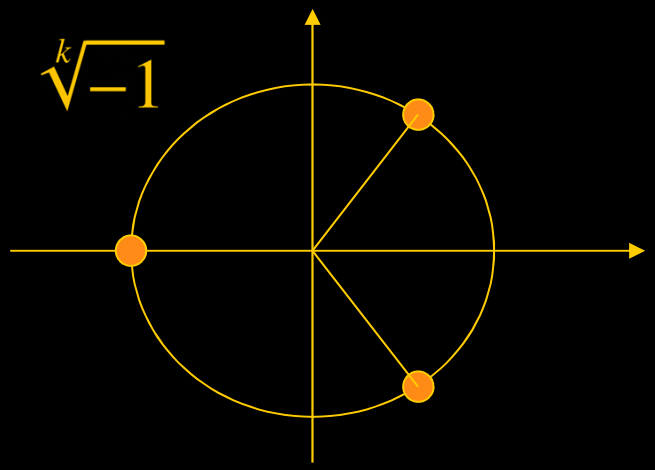
$$\boldsymbol{F} = \boldsymbol{A}^{x}$$

  - MATLAB command

```
F=A^x
```

  - Rotations needed

$$\gamma = \mathrm{e}^{2\pi \mathrm{j}/k}$$



Example 2.7 Cubic Roots

- Given matrix $\boldsymbol{A}$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

- MATLAB code

```
>> A=[1,2,3; 4,5,6; 7,8,0];
   C=A^(1/3), e=norm(A-C^3)
```

- Other two roots

```
>> j1=exp(sqrt(-1)*2*pi/3);
   A1=C*j1, A2=C*j1^2,
   norm(A-A1^3), norm(A-A2^3)
```

Matrix Dot Operations

- Element-by-element operation
  - Dot product `C=A.*B` $c_{ij} = a_{ij}b_{ij}$
  - Dot power `B=A.^A` $b_{ij} = a_{ij}^{a_{ij}}$

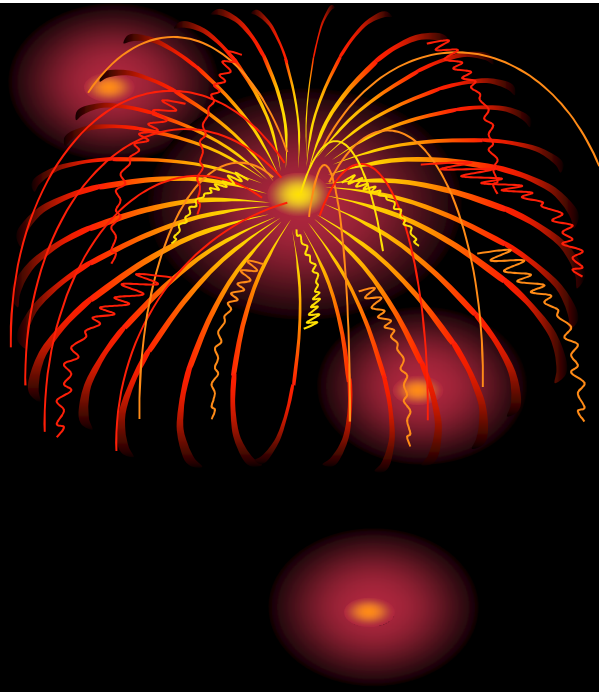
$$\begin{bmatrix} 1^1 & 2^2 & 3^3 \\ 4^4 & 5^5 & 6^6 \\ 7^7 & 8^8 & 0^0 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 27 \\ 256 & 3125 & 46656 \\ 823543 & 16777216 & 1 \end{bmatrix}$$

```
>> A=[1,2,3; 4 5,6; 7,8 0]; B=A.^A
```

  - Curve of a function $y = f(x) = x^2,\ y_i = x_i^2$

```
y=x.^2
```

2.2.2 Logic Operations



- Logical variables
  - For new version of MATLAB
  - Non-zero means logic 1
- Logical operations (element-by-element)
  - "And" operation $A \ \& \ B$
  - "Or" operation $A \mid B$
  - "Not" operation $B = \sim A$
  - Exclusive Or $\mathtt{xor}(A, B)$

2.2.3 Relationship Operations of Matrices

- Allowed comparisons:
  - >, >=, <, <=, ==,~=, find(), all(), any()
- Examples:

```
>> A=[1,2,3; 4 5,6; 7,8 0];
   i=find(A>=5)'
```

```
>> a1=all(A>=5)
   a2=any(A>=5)
```

2.2.4 Simplifications and Conversions

- Function simple() can be used to simplify mathematical formula – old version:

```
s1=simple(s)
[s1,how]=simple(s)
```

  - In new versions, use simplify()
- Other commonly used simplification functions
  - numden(), collect(), expand(), factor()

Example 2.8 Polynomial

□ Find the simplest form of the polynomial

$$P(s) = (s+3)^2(s^2+3s+2)(s^3+12s^2+48s+64)$$

□ Process it with various functions

```
>> syms s;
   P=(s+3)^2*(s^2+3*s+2)*(s^3+12*s^2+48*s+64)
```

In old versions

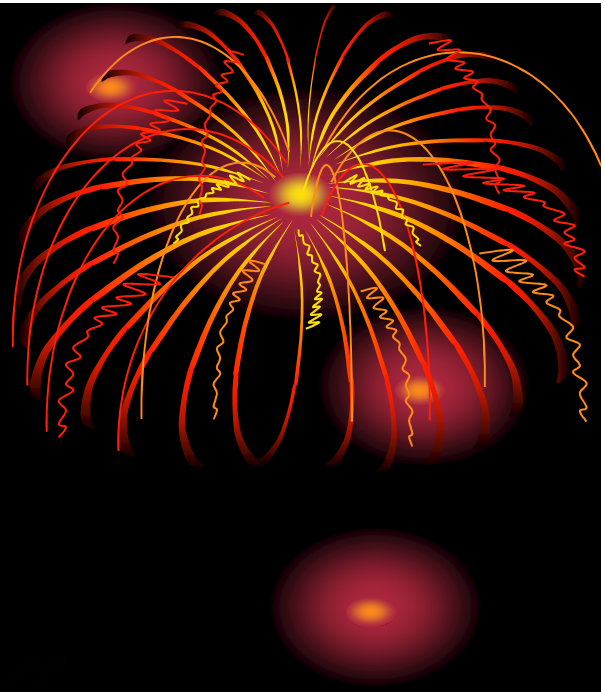
```
>> P1=simple(P)
   [P2,m]=simple(P)
```

□ Simplify the polynomial

```
>> P4=simplify(P)
```

```
>> P3=expand(P)
```

Variable Substitution



- Two commands to use

$$f_1 = \texttt{subs}(f, x_1, x_1^*)$$
$$f_1 = \texttt{subs}(f, \{x_1, x_2, \cdots, x_n\}, \ldots \{x_1^*, x_2^*, \cdots, x_n^*\})$$

- It is run on the dot operation basis
- Convert to LATEX expression

$$f_1 = \texttt{latex}(f)$$

Example 2.9 Math Display

- Function

$$f(t) = \cos(at + b) + \sin(ct)\sin(dt)$$

- Use taylor() to evaluate its Taylor expression and convert the results in LATEX

```
>> syms a b c d t;
   f=cos(a*t+b)+sin(c*t)*sin(d*t);
   f1=taylor(f);
   latex(f1)
```

- By MATLAB

```
\cos \left( b \right) -\sin \left( b \right)
 at+ \left( -1/2\,\cos \left( b \right)
 {a}^{2}+cd \right) {t}^{2}+1/6\,\sin \left( b
 \right) {a}^{3}{t}^{3}+ \left( 1/24\,\cos
 \left( b \right) {a}^{4}-1/6\,c{d}^{3}-
 1/6\,{c}^{3}d \right) {t}^{4}-{\frac
 {1}{120}}\,\sin \left( b \right)
 {a}^{5}{t}^{5}
```

- By LaTeX

$$\cos b - at\sin b + \left(-\frac{a^2\cos b}{2} + cd\right)t^2 + \frac{a^3\sin b}{6}t^3 + \left(\frac{a^4\cos b}{24} - \frac{cd^3}{6} - \frac{c^3 d}{6}\right)t^4 - \frac{a^5\sin b}{120}t^5$$

2.2.5 Basic Discrete Mathematics Computations

Function	Calling format
floor()	$n = \texttt{floor}(x)$
ceil()	$n = \texttt{ceil}(x)$
round()	$n = \texttt{round}(x)$
fix()	$n = \texttt{fix}(x)$
rat()	$[n, d] = \texttt{rat}(x)$
rem()	$B = \texttt{rem}(A, C)$
gcd()	$k = \texttt{gcd}(n, m)$
lcm()	$k = \texttt{lcm}(n, m)$
factor()	$\texttt{factor}(n)$
isprime()	$v_1 = \texttt{isprime}(v)$

Example 2.10 Finding Integers

- Data set
  - -0.2765, 0.5772,1.4597, 2.1091, 1.191,-1.6187
- Observe the results from different rounding functions.

```
>> A=[-0.2765,0.5772,1.4597,...
      2.1091,1.191,-1.6187];
   v1=floor(A), v2=ceil(A)
   v3=round(A), v4=fix(A)
```

Example 2.11 Rationalization

- 3x3 Hilbert matrix can be specified with the statement ***A*** = hilb(3), perform the rational transformation.

```
>> A=hilb(3); [n,d]=rat(A)
```

- result

$$n = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \; d = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}.$$

Example 2.12 GCD/LCM

- Two numbers 1856120 , 1483720
  - get the GCD (greatest common divisor)
  - LCM (least common multiple)
  - prime factor decomposition to the least common multiplier
- Better to use symbolic form

```
>> m=sym(1856120); n=sym(1483720);
   gcd(m,n), lcm(m,n),
   factor(lcm(n,m))
```

Example 2.13 Prime Numbers

- Prime numbers in 1-1000

```
>> A=1:1000; B=A(isprime(A))
```

  - Two statements. Recall C programming

- The prime numbers obtained

2	29	67	107	157	199	257	311	367	421	467	541	599	647	709	769	829	887	967
3	31	71	109	163	211	263	313	373	431	479	547	601	653	719	773	839	907	971
5	37	73	113	167	223	269	317	379	433	487	557	607	659	727	787	853	911	977
7	41	79	127	173	227	271	331	383	439	491	563	613	661	733	797	857	919	983
11	43	83	131	179	229	277	337	389	443	499	569	617	673	739	809	859	929	991
13	47	89	137	181	233	281	347	397	449	503	571	619	677	743	811	863	937	997
17	53	97	139	191	239	283	349	401	457	509	577	631	683	751	821	877	941	
19	59	101	149	193	241	293	353	409	461	521	587	641	691	757	823	881	947	
23	61	103	151	197	251	307	359	419	463	523	593	643	701	761	827	883	953	

2.3 Flow Control Structures of MATLAB Language

- With Control structures, complicated and sophisticated programs can be written
  - Loop control structures
  - Conditional control structures
  - Switch structure
  - Trial structure

2.3.1 Loop Control Structures

- The for loop structures

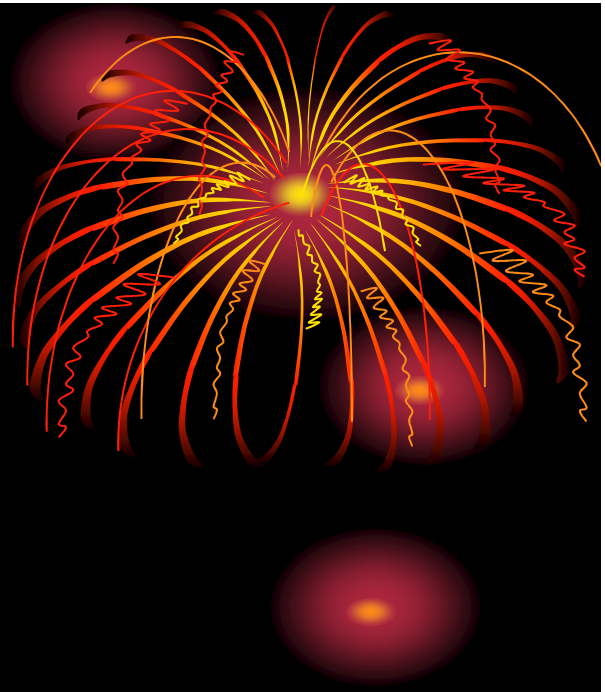
```
for i=v
     loop programs body,
end
```

  - Every term in ***v*** is taken

- If ***v*** is a matrix, *i* pick up one column at a time

```
for (i=0; i<=n-1; i++)
```

While Loops



- The while loop structures

```
while (condition)
    loop structure body,
end
```

- Loops can be nested
  - Different structures have different characteristics
  - Loops can be broken with break command

Example 2.14 Sum of Sequence

- Compute the sum of $\sum_{i=1}^{100} i$ using loop structures.

```
>> s1=0; for i=1:100, s1=s1+i; end
   s2=0; i=1;
   while (i<=100), s2=s2+i; i=i+1;
   end
```

- the simplest statement

```
>> sum(1:100)
```

Example 2.15 How Many Terms

- Find the minimum value of m such that

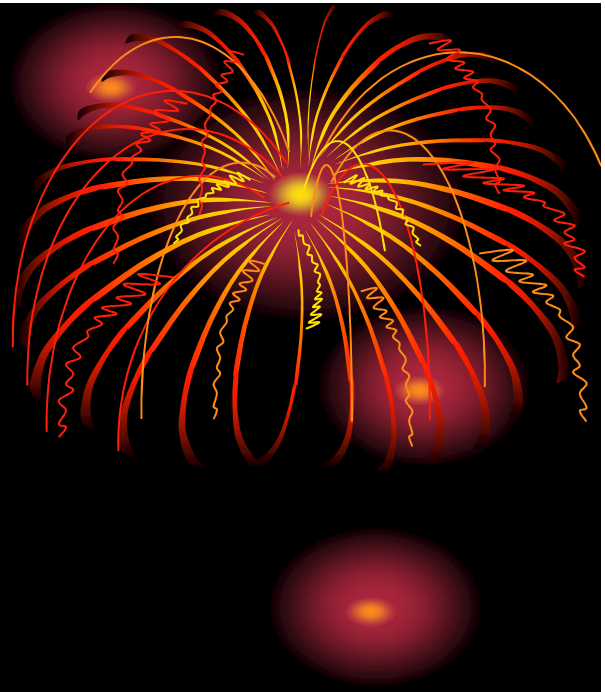
$$\sum_{i=1}^{m} i > 10000$$

- Only while can be used

```
>> s=0; m=0;
   while (s<=10000), m=m+1;
   s=s+m; end, [s,m]
```

- For structure cannot act alone for the problem

Example 2.16 Loops or Vectorization



- Evaluate the sum of the series

$$S = \sum_{i=1}^{100000} \left(\frac{1}{2^i} + \frac{1}{3^i} \right)$$

- MATLAB loop code, and vectorized code

```
>> tic, s=0;
   for i=1:100000, s=s+1/2^i+1/3^i;
   end; toc
   tic, i=1:100000;
   s=sum(1./2.^i+1./3.^i); toc
```

2.3.2 Conditional Control Structures

```
if (condition 1)
    statement group 1
elseif (condition 2)
    statement group 2
        ⋮   ⋮
else
    statement group n + 1
end
```

Example 2.17 Another Way

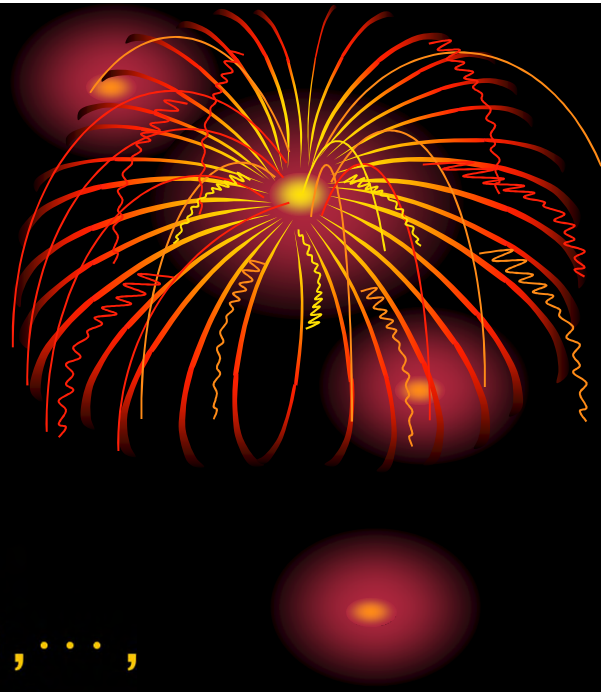
- Using for and if statements to determine the minimum m

$$\sum_{i=1}^{m} i > 10000$$

```
>> s=0;
   for i=1:10000, s=s+i;
       if s>10000, break;
   end, end
```

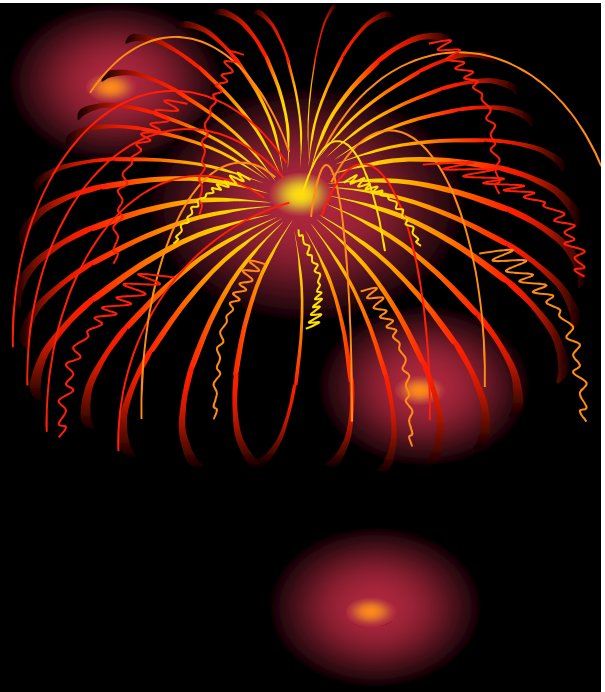
- It is more complicated than the while structure.

2.3.3 Switch Structure



```
switch switch expression
case expression 1, statements 1
case {expression 2, expression 3,⋯,
      expression m}, statements 2
      ⋮
otherwise, statements n
end
```

2.3.4 Trial Structure



- This is a brand new structure

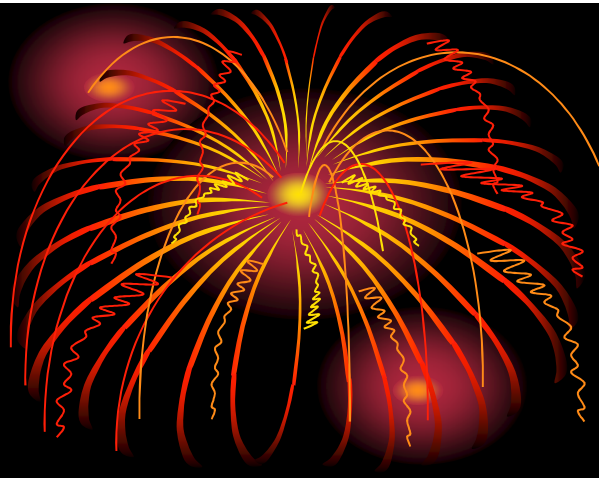
```
try,   statement group 1,
catch,   statement group 2,
end
```

- Advantages:
  - An error trap
  - More efficient algorithm implementation

2.4 Writing and Debugging MATLAB Functions

- Basic structure of MATLAB functions
- Programming of functions with variable inputs/outputs
- Inline functions and anonymous functions

2.4.1 Basic Structure of MATLAB Functions



- Function structures

```
function [return argument list] ...
                =funname(input argument list)
    comments led by %
    input and output variables check
    main body of the function
```

- Important functions
  - nargin, nargout, varargin, varargout

Example 2.18 Why Functions are Needed?

- Problem

$$\sum_{i=1}^{m} i > 10000$$

- M-script can be written and saved as an M-file.
  - If 10000 is changed to other values, the M-file should be modified
- A new file format (function) is needed

Example 2.19 M-Function Implementation

- Write an M-function for Example 2.18
- M-function

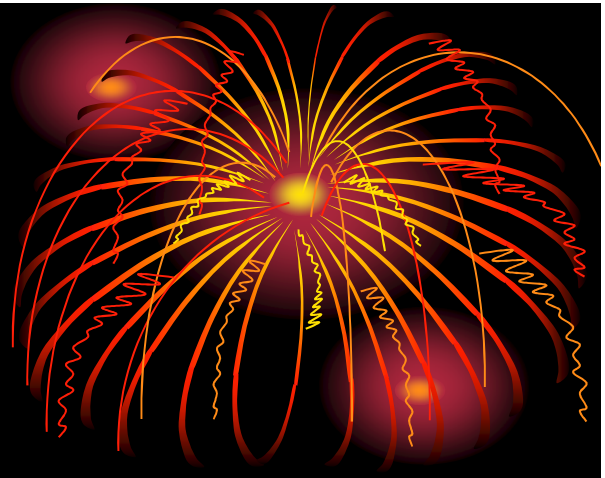
```
function [m,s]=findsum(k)
s=0; m=0;
while (s<=k), m=m+1; s=s+m; end
```

- Example

```
>> [m1,s1]=findsum(145323)
```

- Advantages: no need to modify for *N*

Example 2.20 Hilbert Matrix Generation



- Write an M-function for $n \times m$ Hilbert matrix

$$h_{i,j} = \frac{1}{i + j - 1}$$

- Requirements
  - If only 1 input argument given, generate a square matrix
  - Write suitable help information
  - Check the numbers of inputs and outputs

□ MATLAB function

```
function A=myhilb(n, m)
%MYHILB The function is used to illustrate MATLAB functions.
% A=MYHILB(N, M) generates an NxM Hilbert matrix A;
% A=MYHILB(N) generate and NxN square Hilbert matrix A;
%
%See also: HILB.

% Designed by Professor Dingyu XUE, Northeastern University, PRC
% 5 April, 1995, Last modified by DYX at 30 July, 2001
if nargout>1, error('Too many output arguments.'); end
if nargin==1, m=n; % if one input argument used, square matrix
elseif nargin==0 | nargin>2
   error('Wrong number of iutput arguments.');
end
for i=1:n, for j=1:m, A(i,j)=1/(i+j-1); end, end
```

- On-line help commands

```
>> help myhilb
>> doc myhilb
```

- Generate Hilbert matrices

```
>> A1=myhilb(4,3)
   A2=myhilb(4)
```

- Limitations: cannot generate symbolic matrix
  - Modify kernel statements

```
[i,j]=meshgrid(1:m,1:n); A=1./(i+j-1);
```

```
>> A3=myhilb(sym(4),3)
```

Example 2.21 Recursive Implementation of Factorials

- Recursive relations $n! = n(n-1)!$
- Recursive function implementation

```
function k=my_fact(n)
if nargin~=1,
   error('Error: Only one input variable accepted');
end
if abs(n-floor(n))>eps | n<0
   error('n should be a non-negative integer');
end
if n>1, k=n*my_fact(n-1);
elseif any([0 1]==n), k=1; end
```

Factorial Calculations

- Calculate 11!

```
>> my_fact(11)
```

- Other functions `factorial(n)`

  `prod(1:n)` `gamma(1+n)`

- Symbolic computation

  `gamma(1+sym(n))`

  `factorial(sym(n))`

Example 2.22 Not Suitable for Recursive Implementation

- Implement Fibonacci sequence
  - Fibonacci sequence

$$a_1 = a_2 = 1$$

$$a_k = a_{k-1} + a_{k-2}\ ,\ k = 3, 4, \cdots$$

  - Recursive implementation in MATLAB

```
function a=my_fibo(k)
 if k==1 | k==2, a=1;
 else, a=my_fibo(k-1)+my_fibo(k-2);
 end
```

- The 25th term (takes 3 seconds)

```
>> tic, my_fibo(25), toc
```

- With for loop, 1000th term can be obtained

```
>> tic, a=[1,1];
   for k=3:1000, a(k)=a(k-1)+a(k-2);
   end, toc
```

- Note: not suitable to use recursive structures
- Symbolic computation

```
>> tic, a=sym([1,1]);
   for k=3:100, a(k)=a(k-1)+a(k-2);
   end, toc
```

2.4.2 Handling Variable Input and Returned Arguments

- Functions with variable numbers of arguments
  - In/out arguments: varargin, varargout
- Extracting actual input arguments
  - varargin{1}, varargin{2},…, varargin{n}
- Storage and transfer of the arguments

Example 2.22 Handling Arbitrary Number of Input Arguments

- conv(): calculate product of 2 polynomials
- Handle arbitrary number of polynomials
- MATLAB programming
  - Extract 1 term from varargin at a time

```
function a=convs(varargin), a=1;
for i=1:length(varargin), a=conv(a,varargin{i}); end
```

```
>> P=[1 2 4 0 5]; Q=[1 2];
   F=[1 2 3]; D=convs(P,Q,F), E=conv(conv(P,Q),F)
   G=convs(P,Q,F,[1,1],[1,3],[1,1])
```

2.4.3 Inline and Anonymous Functions

- inline function (not recommended)
  - No need to create an M-file

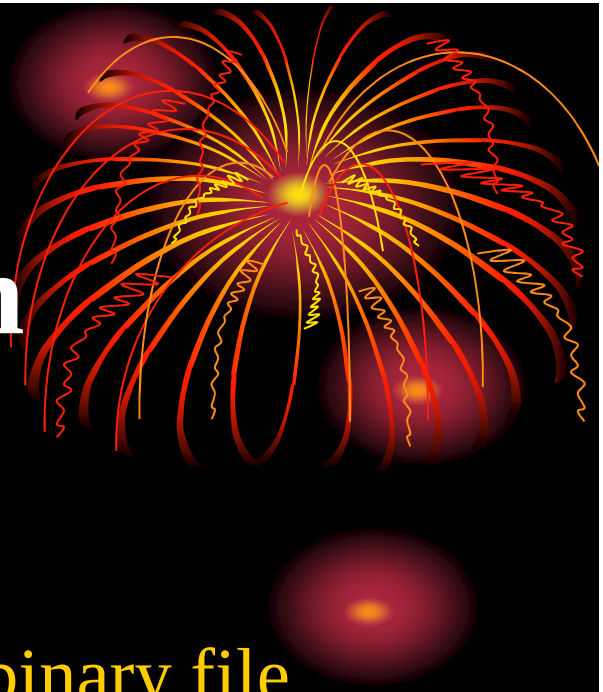
```
fun = inline(function expression,...
             list of variables)
```

- For MATLAB7.0+, recommend anonymous
  - Direct use of variables in MATLAB workspace

```
f = @(list of variables)function_contents
```

- Other types of functions – private, overload

2.4.4 Pseudo Code and Source Code Protection



- Why pseudo code?
  - Increase execution speed
  - Protect source code: ASCII file to binary file
- Creating pseudo code

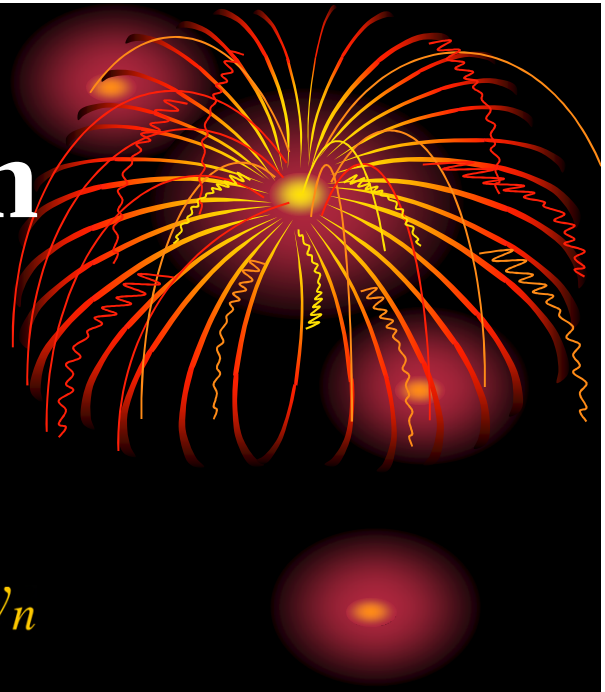
```
pcode mytest  pcode *.m
pcode mytest -inplace
```

- Must reserve source code in safe place
- .p files not reversible

2.5 Two-dimensional Graphics

- Scientific visualization is important in scientific research. Convert data into graphs
  - Basic statements in 2D graphics
  - Plots with multiple vertical axes
  - Other 2D graphics facilities
  - Plotting implicit functions
  - Access data in files (*.txt or *.xls)

2.5.1 Basic Statements in 2D Plots



- Two sequences

$$t = t_1, t_2, \cdots, t_n \qquad y = y_1, y_2, \cdots, y_n$$

  - Construct vectors

$$\boldsymbol{t} = [t_1, t_2, \cdots, t_n]$$

$$\boldsymbol{y} = [y_1, y_2, \cdots, y_n]$$

- Use the data to draw plots

```
plot(t,y)
```

Example 2.24 Function Plotting and Validations

- Draw plot for function

$$y = \sin(\tan x) - \tan(\sin x), \quad x \in [-\pi, \pi]$$

- MATLAB code

```
>> x=[-pi : 0.05: pi];
   y=sin(tan(x))-tan(sin(x)); plot(x,y)
```

- Problem: how to validate curves?
  - Select different step sizes

Variable Step-size Vectors

- Use different step-sizes
  - Use the same small step-size

```
>> x=[-pi:0.00001:pi];
   y=sin(tan(x))-tan(sin(x)); plot(x,y)
```

- Use variable step-size, small around $\pm\pi/2$

```
>> x=[-pi:0.05:-1.8,-1.801:.001:-1.2,...
      -1.2:0.05:1.2,1.201:0.001:1.8,...
      1.81:0.05:pi];
   y=sin(tan(x))-tan(sin(x)); plot(x,y)
```

Example 2.25 Piecewise Functions

- Draw saturation function

$$y = \begin{cases} 1.1\text{sign}(x), & |x| > 1.1 \\ x, & |x| \le 1.1 \end{cases}$$

- MATLAB draw (mutual exclusive conditions)

互斥

```
>> x=[-2:0.02:2];
   y=1.1*sign(x).*(abs(x)>1.1)+...
     x.*(abs(x)<=1.1); plot(x,y)
```

- Simpler statement: draw poly-lines

```
>> plot([-2,-1.1,1.1,2],[-1.1,-1.1,1.1,1.1])
```

Other Syntaxes

- t is vector, while y is a matrix, e.g.,

$$y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

- t and y are matrices, size of t and y the same
- More pairs of vectors or matrices

$$(t_1, y_1), (t_2, y_2), \cdots, (t_m, y_m),$$

$$\texttt{plot}(t_1, y_1, t_2, y_2, \cdots, t_m, y_m)$$

A More General Syntax

- Change the properties directly

h = plot(t_1, y_1, option 1, t_2, y_2, option 2, ⋯, t_m, y_m, option m)

line type	colour		marks	
'-'	'b'	'c'	'*'	'pentagram'
'--'	'g'	'k'	'.'	'o'
':'	'm'	'r'	'x'	'square'
'-.'	'w'	'y'	'v'	'diamond'
'none'			'^'	'hexagram'
			'>'	'<'

Object Properties Extraction and Settings

- Figure window, curve, axes are objects
- Object properties can be assigned with `set()`
- Properties can be extracted with `get()`

```
set(handle,'p_name 1',...
    p_value 1,'p_name 2',p_value 2, ···)
v = get(object, 'p_name')
```

- Object properties can be set with menus

2.5.2 Plots with Multiple Vertical Axes

- Sometimes the differences in the two plots are huge, plotyy() function should be used
- Example 2-26: Draw the two functions

$$y_1 = \sin x,\ y_2 = 0.01\cos x$$

```
>> x=0:0.01:2*pi; y1=sin(x);
   y2=0.01*cos(x); plot(x,y1,x,y2,'--')
```

- More plotting functions for more axes
  - plotyyy(), plot4y(), plotxx() functions
  - Downloadable from MATLAB File Exchange

2.5.3 Drawing Other Forms of 2D Graphics

- Different functions to use

bar(x,y)	comet(x,y)
compass(x,y)	errorbar(x,y,y_m,y_M)
feather(x,y)	fill(x,y,c)
hist(y,n)	loglog(x,y)
polar(x,y)	quiver(x,y)
stairs(x,y)	stem(x,y)
semilogx(x,y)	semilogy(x,y)

Example 2.27 Polar Plots

- Draw polar function curves

$$\rho = 5\sin(4\theta/3),\ \ \rho = 5\sin(\theta/3)$$

- Drawing curves

```
>> theta=0:0.01:2*pi; rho=5*sin(4*theta/3);
   polar(theta,rho)
```

  - Incomplete – find the period

```
>> rho=5*sin(theta/3);
   polar(theta,rho)
```

Drawing plots as Subplots

- Divide graphics window into several parts

  `subplot(`*n*`,`*m*`,`*k*`)`

  - *n* and *m* are rows and columns
  - *k* is the serial number of the portion of window
- Also setting with menu items
  - Add an axis
  - Drag mouse button to adjust axis size

Example 2.27 Different 2D Plots as Subplots in Windows

- Sinusoidal function is used as an example
- Divide graph window as 2x2
- Draw in different portions

```
>> t=0:.2:2*pi; y=sin(t);
   subplot(2,2,1), stairs(t,y)
   subplot(2,2,2), stem(t,y)
   subplot(2,2,3), bar(t,y)
   subplot(2,2,4), semilogx(t,y)
```

2.5.4 Drawing Implicit Functions

- Example of an implicit function

$$x^2 \sin(x + y^2) + y^2 e^{x+y} + 5\cos(x^2 + y) = 0$$

- Drawing implicit functions

```
ezplot(implicit function expression)
```

  - Represent implicit functions as strings
  - Default region is $[-2\pi, 2\pi]$

- Other syntaxes `ezplot(im_function,[`x_m`,`x_M`])`

`ezplot(im_function,[`x_m`,`x_M`,`y_m`,`y_M`])`

Example 2.29 Implicit Function

- Please draw

$$x^2 \sin(x + y^2) + y^2 e^{x+y} + 5\cos(x^2 + y) = 0$$

- MATLAB statements -- handles

```
>> h=ezplot('x^2 *sin(x+y^2) +...
            y^2*exp(x+y)+5*cos(x^2+y)');
   set(h,'Color','b')
```

- Over a large region

```
>> h=ezplot('x^2 *sin(x+y^2) +y^2*exp(x...
          +y)+5*cos(x^2+y)',[-10 10]);
   set(h,'Color','b')
```

2.5.5 Decoration of Plots

- Direct use of toolbar
- Text decorations
  - Special symbols LaTeX
  - Subscript and superscripts, with ^ and _

```
a_2^2+b_2^2=c_2^2
```

$$a_2^2 + b_2^2 = c_2^2$$

  - LaTeX advantages. Better to use overpic package
- New functions since MATLAB 7.0

2.5.6 Access Data Files

- Commands save and load

```
save mydat A B C
save /ascii mydat.dat A B C
X = load(filename)
```

- Exchange between MATLAB & Excel

```
X = xlsread(filename,range)
```

```
'B6:C67'
```

  - Write file: xlswrite()

Example 2-30 Access Excel File

- Known Excel file: census.xls – population
  - Open file  `>> open('census.xls')`
  - Rows 5-67 stores data
  - B column stores year, C columns, population
- Read into MATLAB, the plotting

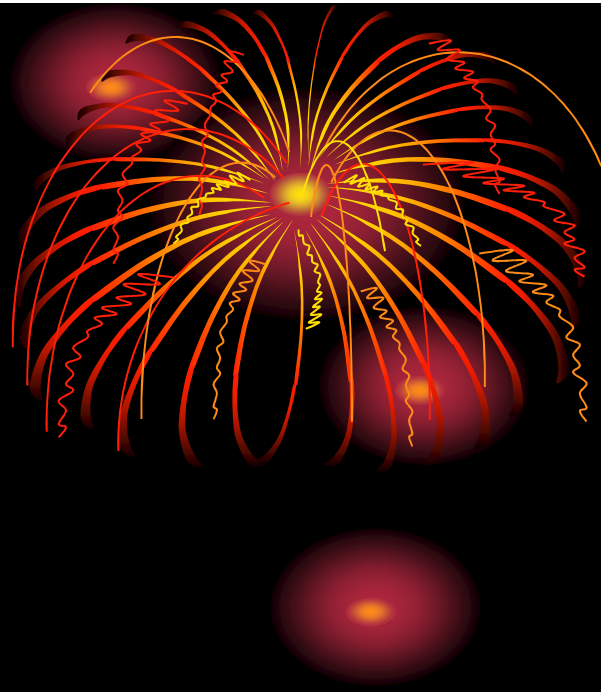
```
>> X=xlsread('census.xls','B5:C67');
   t=X(:,1); p=X(:,2);
   plot(t,p)
```

- Simpler method: Copy & Paste

2.6 Three-dimensional Graphics

- Difficult to obtain with other computer languages, easy to obtain with MATLAB
- Main topics in the section
  - 3D curves
  - 3D surfaces
  - View-point setting
  - 3D implicit function plots
  - Rotating 3D graphs

2.6.1 3D Curves



- Plotting 3D curves

  plot3(x, y, z)

  plot3(x_1, y_1, z_1, option 1,...
  x_2, y_2, z_2, option 2,...
  x_m, y_m, z_m, option m)

- Other 3D plotting functions
  - stem3, fill3, bar3
  - Draw trajectory dynamically: comet3

Example 2.31 Position of a Particle in 3D Space

- 3D parametric equation

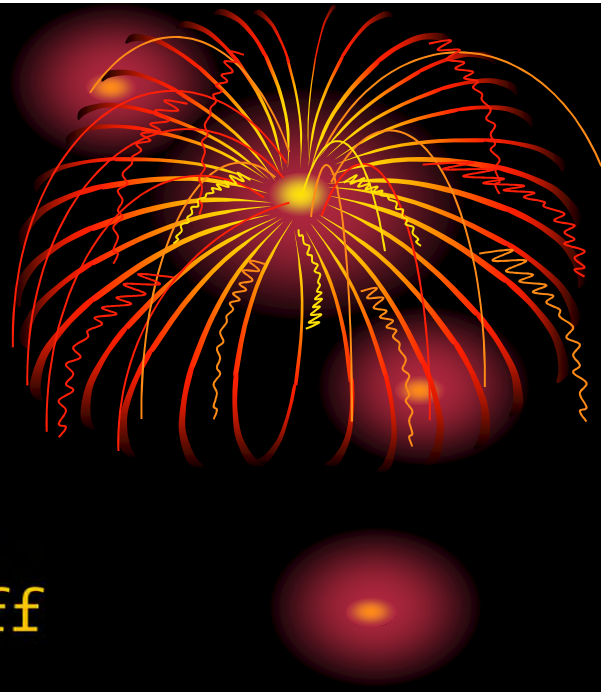
$$x(t) = t^3 \sin(3t)\mathrm{e}^{-t}, y(t) = t^3 \cos(3t)\mathrm{e}^{-t}, z = t^2$$

  - Position of a particle (change with time)
  - where, $t \in [0, 2\pi]$

- MATLAB plotting (dot calculation)

```
>> t=0:0.01:2*pi; x=t.^3.*sin(3*t).*exp(-t);
   y=t.^3.*cos(3*t).*exp(-t);
   z=t.^2; plot3(x,y,z), grid
```

Other 3D Plots



- With function `stem3()`

```
>> stem3(x,y,z); hold on;
   plot3(x,y,z), grid; hold off
```

- Dynamic trajectory

```
>> figure; comet3(x,y,z)
```

- Toolbar usages
  - View point adjustment in 3D plots
  - Read coordinate, zooming facilities

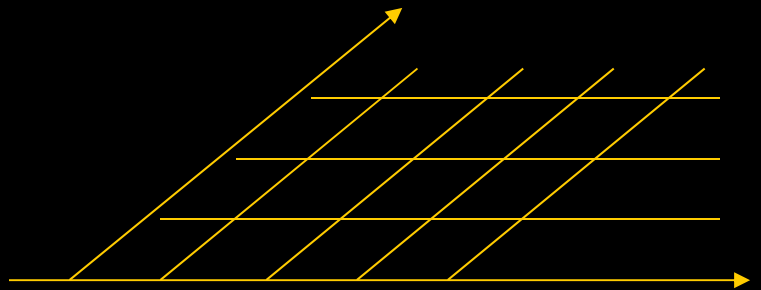
2.6.2 3D Surface Plots

- Draw surfaces

  `[x,y]=meshgrid(v1, v2)`

  `z= ..., for instance z=x.*y`

  `surf(x,y,z) or mesh(x,y,z)`



- Other functions
  - surfl(), surfc()
- Contour plots
  - contour(), contours()

Example 2.32 3D Surface

- Given function with 2 variables

$$z = f(x, y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$$

- MATLAB plot

```
>> [x,y]=meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   mesh(x,y,z)
```

- Surface plot

```
>> surf(x,y,z)
```

Example 2.33 Another Function

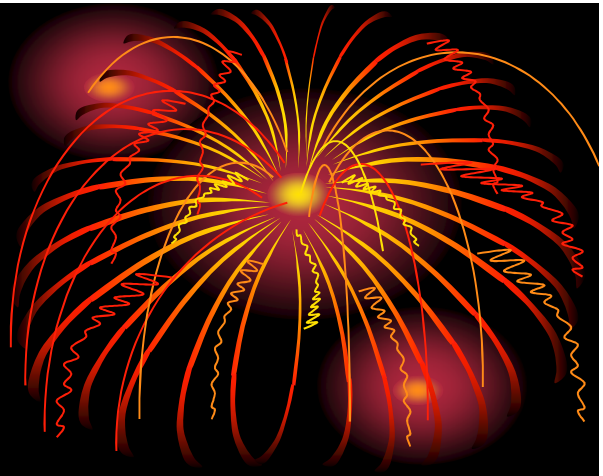
- Two-variable function

$$z = f(x,y) = \frac{1}{\sqrt{(1-x)^2+y^2}} + \frac{1}{\sqrt{(1+x)^2+y^2}}$$

- Draw 3D surface

```
>> [x,y]=meshgrid(-2:.1:2);
   z=1./(sqrt((1-x).^2+y.^2))+...
        1./(sqrt((1+x).^2+y.^2));
   surf(x,y,z), shading flat
```

Step-size Selection



- With variable step-sizes

```
>> xx=[-2:.1:-1.2,-1.1:0.02:-0.9,...
       -0.8:0.1:0.8,0.9:0.02:1.1, 1.2:0.1:2];
   yy=[-1:0.1:-0.2, -0.1:0.02:0.1, 0.2:.1:1];
   [x,y]=meshgrid(xx,yy);
   z=1./(sqrt((1-x).^2+y.^2))+...
       1./(sqrt((1+x).^2+y.^2));
   surf(x,y,z), shading flat;
   zlim([0,15])
```

- Singular point, to be discussed in Ch5

Example 2.34 Handling Piecewise Functions

□ Draw surface for piecewise function

$$p(x_1, x_2) = \begin{cases} 0.5457 \exp(-0.75x_2^2 - 3.75x_1^2 - 1.5x_1), x_1 + x_2 > 1 \\ 0.7575 \exp(-x_2^2 - 6x_1^2), -1 < x_1 + x_2 \le 1 \\ 0.5457 \exp(-0.75x_2^2 - 3.75x_1^2 + 1.5x_1), x_1 + x_2 \le -1. \end{cases}$$

➢ Evaluation of piecewise functions

✓ Mutually exclusive relations, dot operations

✓ Loops, rather complicated

Drawing Piecewise Surfaces

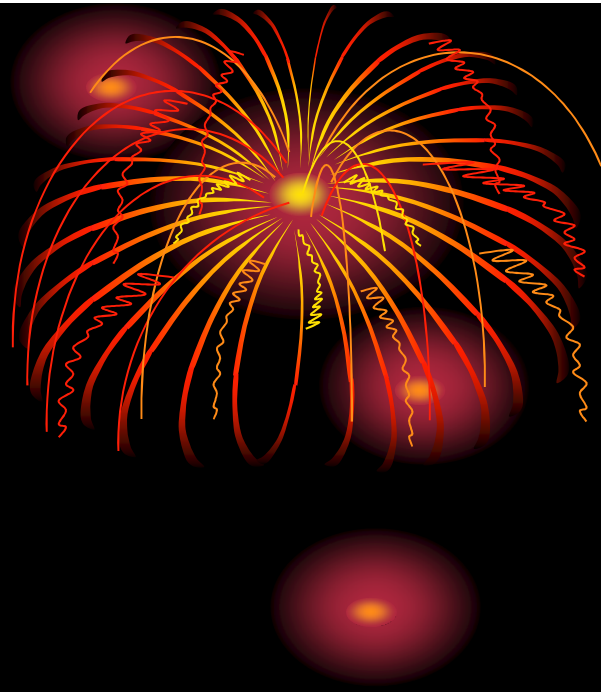
- Math form

$$p(x_1, x_2) = \begin{cases} 0.5457\exp(-0.75x_2^2 - 3.75x_1^2 - 1.5x_1), & x_1 + x_2 > 1 \\ 0.7575\exp(-x_2^2 - 6x_1^2), & -1 < x_1 + x_2 \le 1 \\ 0.5457\exp(-0.75x_2^2 - 3.75x_1^2 + 1.5x_1), & x_1 + x_2 \le -1. \end{cases}$$

- MATLAB plotting
  - Generate mesh grid, evaluate then plot

```
>> [x1,x2]=meshgrid(-1.5:.1:1.5,-2:.1:2);
   p=0.5457*exp(-0.75*x2.^2-3.75*x1.^2-1.5*x1).*(x1+x2>1)+...
     0.7575*exp(-x2.^2-6*x1.^2).*((x1+x2>-1)& (x1+x2<=1))+...
     0.5457*exp(-0.75*x2.^2-3.75*x1.^2+1.5*x1).*(x1+x2<=-1);
   surf(x1,x2,p), xlim([-1.5 1.5]);
```

2.6.3 Contour Plots



- Various of contour plots
  - Direct draw `contour(x,y,z,n)`
  - Contours with labels

```
[C,h] = contour(x,y,z,n)
clabel(C,h)
```

  - 3D contours

`contour3(x,y,z,n)` `contourf(x,y,z,n)`

Example 2-35 Contours

- Example 2-34, piecewise function
  - Generate data, draw contours

```
>> [x1,x2]=meshgrid(-1.5:.1:1.5,-2:.1:2);
   p=0.5457*exp(-0.75*x2.^2-3.75*x1.^2-1.5*x1).*(x1+x2>1)+...
     0.7575*exp(-x2.^2-6*x1.^2).*((x1+x2>-1)& (x1+x2<=1))+...
     0.5457*exp(-0.75*x2.^2-3.75*x1.^2+1.5*x1).*(x1+x2<=-1);
   [C,h]=contour(x1,x2,p); clabel(C,h)
```

- 3D contours

```
>> contourf(x1,x2,p);
   figure; contour3(x1,x2,p,30)
```

2.6.4 Plots of 3D Implicit Functions

- 3D implicit function $f(x, y, z) = 0$
- Download ezimplot3() from File Exchange

  `ezimplot3(fun,[`$x_m, x_M, y_m, y_M, z_m, z_M$`])`

  - Default region [-2π,2π]
- Fun can be of
  - Implicit function strings
  - Anonymous functions
  - M-functions

Example 2-36 3D Implicit Function Plots

- Given 3D implicit functions

$$x(x,y,z) = x\sin\left(y+z^2\right) + y^2\cos\left(x+z\right)$$
$$+zx\cos\left(z+y^2\right) = 0$$

- Drawing statements

```
>> f='x*sin(y+z^2)+y^2*cos(x+z)+...
      z*x*cos(z+y^2)';
   f=@(x,y,z)x*sin(y+z^2)+y^2*cos(x+z)+...
      z*x*cos(z+y^2);
>> ezimplot3(f,[-1 1]);
>> f1='x^2+y^2+z^2-1'; ezimplot3(f1,[-1 1]);
```

2.6.5 View-point Specifications

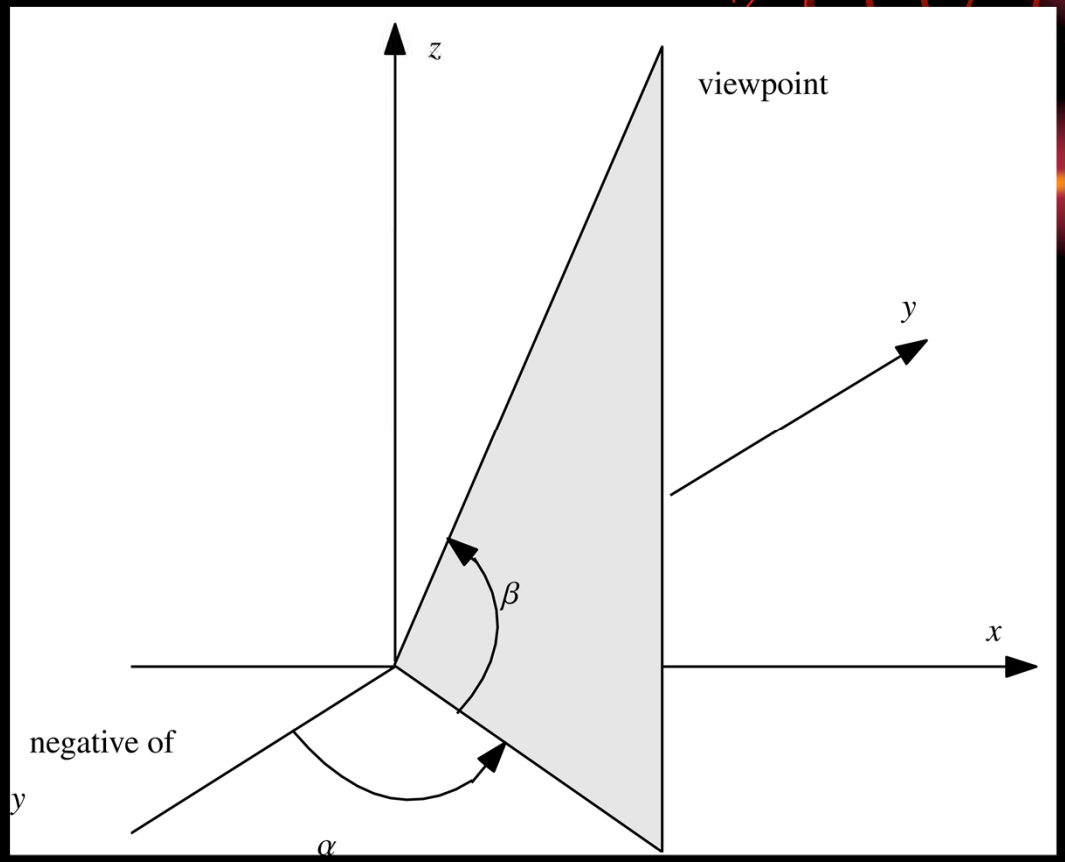
- Two ways of setting view points
  - Direct use of toolbar
  - Command `view()`

    `view(`α`,`β`)`
  - Read current view points `[`α`,`β`]=view(3)`
- α is azimuth angle, β is elevation angle
- Uniquely define the view-point

Definition of View-points

- Settings

  `view(`α, β`)`



Example 2.37 Orthographic Views of 3D Plots

三视图

- Given function

$$z = f(x,y) = (x^2 - 2x)e^{-x^2-y^2-xy}$$

- Default angle extraction `>> [a b]=view(3);`
- 3D surface plot

```
>> [x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   subplot(224), surf(x,y,z),
   subplot(221), surf(x,y,z), view(0,90);
   subplot(222), surf(x,y,z), view(90,0);
   subplot(223), surf(x,y,z), view(0,0);
```

2.6.6 Rotations of 3D Surfaces

- Rotating function

  `rotate(h,`v`,`α`)`

  - where
    - ✓Handles of 3D plot h
    - ✓Rotate baseline by vector $\boldsymbol{v}$
    - ✓Angle is α
  - Example: rotate along positive direction of x axis, $\boldsymbol{v}$ = [1,0,0]

Example 2-38 Rotating 3D Plots

- Rotate the piecewise plot

```
>> [x,y]=meshgrid(-1:.04:1,-2:.04:2);
   z= 0.5457*exp(-0.75*y.^2-3.75*x.^2-1.5*x).*(x+y>1)+...
      0.7575*exp(-y.^2-6*x.^2).*((x+y>-1) & (x+y<=1))+...
      0.5457*exp(-0.75*y.^2-3.75*x.^2+1.5*x).*(x+y<=-1);
   h=surf(x,y,z);
```

- Rotate along *x* axis, baseline *v*=[1,0,0]

```
>> rot_ax=[1,0,0]; rotate(h,rot_ax,15)
```

- Rotate along baseline *v* =[1 1 1]

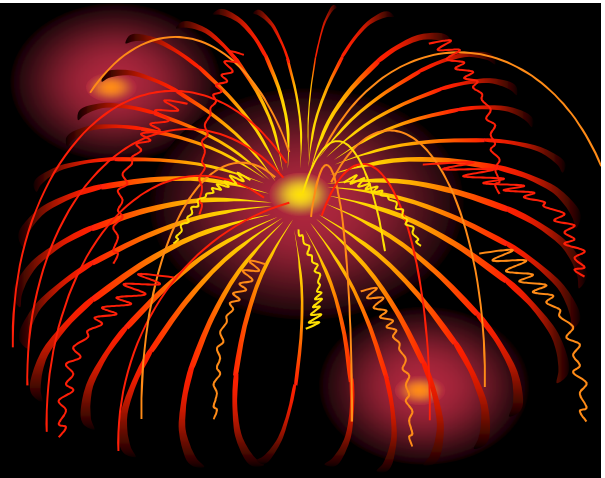
```
>> h=surf(x,y,z); rot_ax=[1,1,1];
   rotate(h,rot_ax,15)
```

Animation of Rotations

- Rotate along *x* axis for 360 degrees
  - Rotate 1 degree per step (0.01s pause)
  - Loop structure needed

```
>> figure; h=surf(x,y,z);
   r_ax=[1 0 0]; axis tight
   for i=0:360,
      rotate(h,r_ax,1); pause(0.01),
   end
```

2.7 Drawing 4D Plots



- 4D plots
  - Time-driven 3D animation
  - Slice view of 3D plots: volume visualization
  - Examples
    - ✓CT images, temperature or density of a 3D solid
    - ✓Slices are needed to view inside
    - ✓Flow rate and concentration of liquid
  - MATLAB function

$$\texttt{slice}(x, y, z, V, x_1, y_1, z_1)$$

体视化

Example 2-39 Slice Views of 3D Solid Objects

- Function

$$V(x,y,z)=\sqrt{x^x+y^{(x+y)/2}+z^{(x+y+z)/3}}$$

  - Ordinary slices
    - ✓Generate 3D mesh grids
    - ✓Calculate function values in the grids (dot operation)
    - ✓Plotting

```
>> [x,y,z]=meshgrid(0:0.1:2);
   V=sqrt(x.^x+y.^((x+y)/2)+z.^((x+y+z)/3));
   slice(x,y,z,V,[1 2],[1 2],[0 1]);
```

Special Slices

- Construct a plane at $z = 1$
- Rotate 45^o the plane along x axis
- Slice with the plane

```
>> [x0,y0]=meshgrid(0:0.1:2);
   z0=ones(size(x0));
   h=surf(x0,y0,z0); rotate(h,[1,0,0],45);
   x1=get(h,'XData'); y1=get(h,'YData');
   z1=get(h,'ZData');
   slice(x,y,z,V,x1,y1,z1),
   hold on, slice(x,y,z,V,2,2,0)
```

A Volume Visualization Interface

- A volume visualization interface is developed vol_visual4d()
- Syntax
  - Generate volume data x, y, z, V
  - Call the function

    `vol_visual4d(`x`,`y`,`z`,`V`)`
  - Freely setting different slices

Example 2-40 Use the Volume Visualization Interface

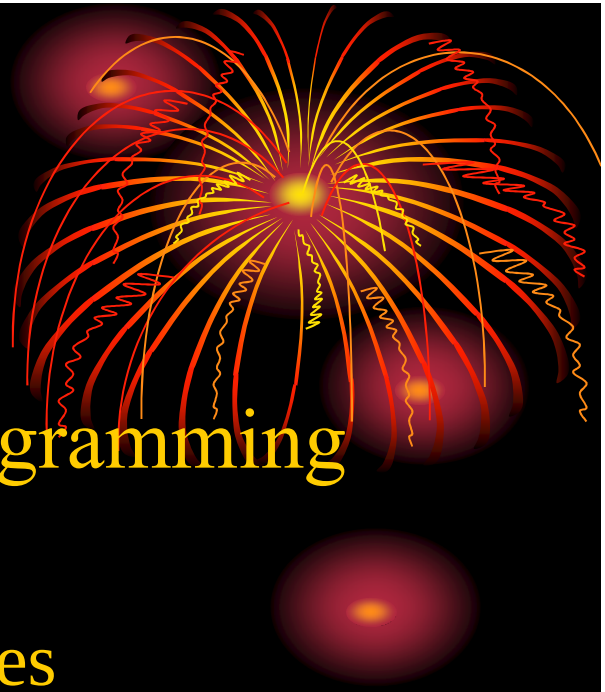
- Function

$$V(x,y,z) = \sqrt{x^x + y^{(x+y)/2} + z^{(x+y+z)/3}}$$

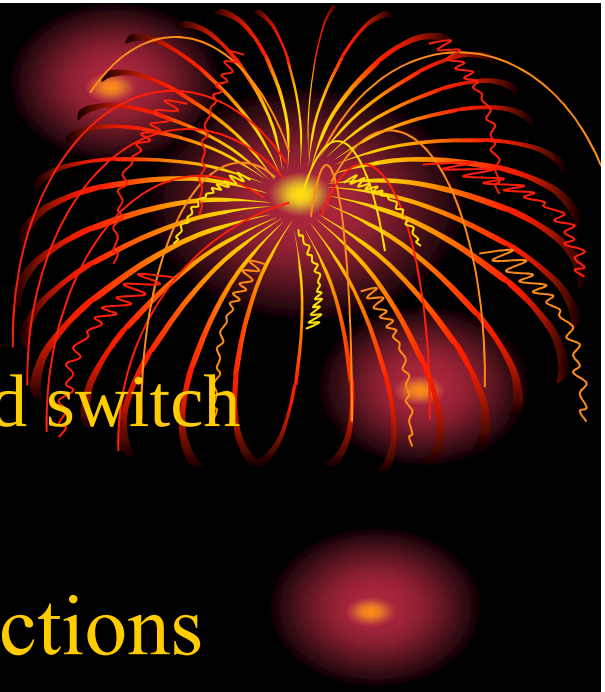
- Slice plotting
  - Use scrollbars to adjust the slices
  - Select whether a slice is shown

```
>> [x,y,z]=meshgrid(0:0.1:2);
   V=sqrt(x.^x+y.^((x+y)/2)+z.^((x+y+z)/3));
   vol_visual4d(x,y,z,V);
```

Chapter Summary



- Fundamentals of MATLAB programming
  - Constants and variables
  - Data types and statement structures
  - Colon expression and sub-matrix extraction
- Matrix computation
  - Algebraic, logic and relational operations
  - Expression simplification and conversion
  - Some discrete math computation



- Control flow structures
  - Two kinds of loops, conditional and switch
  - Trial structure in MATLAB
- Main stream programming -- functions
  - Standard function structures
  - Examming input and output arguments
  - With arbitrary number of in/out arguments
  - Recursive structures
  - Pseudo-code manipulations

- Two-dimensional graphics
  - Descartes, polar, logarithmic and bars
  - Implicit function plotting
  - Decorations of plots
  - View point setting
- Three-dimensional plots and surfaces
  - 3D implicit functions
  - Rotations of plots
- Four-dimensional volume visualization
  - An easy-to-use GUI